Inheritance in C++

- Introduction
- Inheritance: Base Classes and Derived Classes
- Protected Members
- Casting Base-Class Pointers to Derived-Class Pointers
- Using Member Functions
- Overriding Base-Class Members in a Derived Class
- Public, Protected and Private Inheritance
- Direct Base Classes and Indirect Base Classes
- Using Constructors and Destructors in Derived Classes
- Implicit Derived-Class Object to Base-Class Object Conversion
- Software Engineering with Inheritance
- Composition vs. Inheritance
- "Uses A" and "Knows A" Relationships
- Case Study: Point, Circle, Cylinder

Introduction

- Inheritance
  - New classes created from existing classes
  - Absorb attributes and behaviors.
- Polymorphism
  - Write programs in a general fashion
  - Handle a wide variety of existing (and unspecified) related classes
- Derived class
  - Class that inherits data members and member functions from a previously defined base class

Base and Derived Classes

- Often an object from a derived class (subclass) "is an" object of a base class (superclass)

<table>
<thead>
<tr>
<th>Base class</th>
<th>Derived classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>GraduateStudent</td>
</tr>
<tr>
<td></td>
<td>UndergraduateStudent</td>
</tr>
<tr>
<td>Shape</td>
<td>Circle</td>
</tr>
<tr>
<td></td>
<td>Triangle</td>
</tr>
<tr>
<td></td>
<td>Rectangle</td>
</tr>
<tr>
<td>Loan</td>
<td>CarLoan</td>
</tr>
<tr>
<td></td>
<td>HomeImprovementLoan</td>
</tr>
<tr>
<td></td>
<td>MortgageLoan</td>
</tr>
<tr>
<td>Employee</td>
<td>FacultyMember</td>
</tr>
<tr>
<td></td>
<td>StaffMember</td>
</tr>
<tr>
<td>Account</td>
<td>CheckingAccount</td>
</tr>
<tr>
<td></td>
<td>SavingsAccount</td>
</tr>
</tbody>
</table>

Implementation of public inheritance

```cpp
class CommissionWorker : public Employee {
    ...
};
```

Class CommissionWorker inherits from class Employee
- friend functions not inherited
- private members of base class not accessible from derived class
protected members

- protected inheritance
  - Intermediate level of protection between public and private inheritance
  - Derived-class members can refer to public and protected members of the base class simply by using the member names
  - Note that protected data "breaks" encapsulation

Object of a derived class
- Can be treated as an object of the base class
- Reverse not true - base class objects not a derived-class object

Downcasting a pointer
- Use an explicit cast to convert a base-class pointer to a derived-class pointer
- Be sure that the type of the pointer matches the type of object to which the pointer points

derivedPtr = static_cast< DerivedClass * > basePtr;

Casting Base Class Pointers to Derived Class Pointers

- Example
  - Circle class derived from the Point base class
  - We use pointer of type Point to reference a Circle object, and vice-versa

Casting Base-Class Pointers to Derived-Class Pointers

- Circle class definition
  - We use pointer of type Point to reference a Circle object, and vice-versa

Circle definition

- Class Circle derived from Point
- We use pointer of type Point to reference a Circle object, and vice-versa

1. Circle definition
### finitions

1. **Load headers**
   - `#include <iostream>`
   - `using std::cout;`
   - `using std::endl;`
   - `#include <iomanip>`

2. **Include headers**
   - `#include "point.h"`
   - `#include "circle.h"`

3. **Function Definitions**
   - **Driver**
     - **1. Initialize objects**
       - **Load headers**
       - **1.1 Initialize objects**

### Program Output

```
Point p: [30, 50]  
Circle c: Center = [120, 89]; Radius = 2.70
Circle c (via *pointPtr):  
Circle c (via *circlePtr):  
Area of c (via circlePtr): 22.90
Point p (via *circlePtr):  
Area of object circlePtr points to: 0.00
```

### Using Member Functions

- **Derived class**
  - Cannot directly access `private` members of its base class
  - Hiding `private` members is a huge help in testing, debugging and correctly modifying systems

### Overriding Base-Class Members in a Derived Class

- **To override a base-class member function**
  - In derived class, supply new version of that function
    - Same function name, different definition
  - The scope-resolution operator may be used to access the base class version from the derived class

```c++
// employ.h
// Definition of the class Employee
#ifndef EMPLOY_H
#define EMPLOY_H

class Employee {
public:
  Employee( const char *, const char * );  // constructor
  void print() const ; // output first and last name
  ~Employee();        // destructor
private:
  char *firstName;    // dynamically allocated string
  char *lastName;    // dynamically allocated string
};

#define EMPLOY_H
```

```c++
// employ.cpp
// Member function definitions for class Employee
#include <iostream>

using std::cout;

#include <cstring>
#include <cassert>
#include "employ.h"

Employee::Employee( const char *first, const char *last )
{
  firstName = new char [ strlen( first ) + 1 ];
  strcpy( firstName, first );
  lastName = new char [ strlen( last ) + 1 ];
  strcpy( lastName, last );
}
```

### 1. Employee class definition

1. **Load header**

2. **1.1 Function definitions**

```c++
void print();  // output first and last name
```
Bob Smith is an hourly worker with pay of $400.00.

```cpp
int main()
{
    #include "hourly.h"

    Employee::print();   // call base-class print function
    cout << "HourlyWorker::print() is executing

    HourlyWorker h( "Bob", "Smith", 40.0, 10.00 );
    h.print();

    double wage = initWage;    // should validate
    hours = initHours;  // should validate
    double initHours, double initWage )

    double HourlyWorker::getPay() const { return wage * hours; }
}
```

### Direct and Indirect Base Classes

- **Direct base class**
  - Explicitly listed derived class' header with the colon (:) notation when that derived class is declared.
  - `class HourlyWorker : public Employee` is a direct base class of `HourlyWorker`

- **Indirect base class**
  - Inherited from two or more levels up the class hierarchy
  - `class MinuteWorker : public HourlyWorker` is an indirect base class of `MinuteWorker`

### Using Constructors and Destructors in Derived Classes

- **Base class initializer**
  - Uses member-initializer syntax
  - Can be provided in the derived class constructor to call the base-class constructor explicitly
    - Otherwise base class' default constructor called implicitly
  - Base-class constructors and base-class assignment operators are not inherited by derived classes
    - However, derived-class constructors and assignment operators can call them
Using Constructors and Destructors in Derived Classes

- Derived-class constructor
  - Calls the constructor for its base class first to initialize its base-class members
  - If the derived-class constructor is omitted, its default constructor calls the base-class' default constructor
- Destructors are called in the reverse order of constructor calls.
  - Derived-class destructor is called before its base-class destructor

---

1. Point definition
1. Load header
1.1 Function definitions

---

1. Circle Definition
1.1 Function definitions

---

1. Initialize objects
2. Objects enter and leave scope

---

Program Output

Point constructor: [11, 22]
Point destructor: [11, 22]

Point constructor: [72, 29]
Circle constructor: radius is 4.5 [72, 29]
Point constructor: [5, 5]
Circle constructor: radius is 10 [5, 5]
Circle constructor: radius is 4.5 [72, 29]
Point constructor: [72, 29]
Implicit Derived-Class Object to Base-Class Object Conversion

- `baseClassObject = derivedClassObject;`
  - This will work
    - Remember, the derived class object has more members than the base class object
  - Extra data is not given to the base class
- `derivedClassObject = baseClassObject;`
  - May not work properly
    - Unless an assignment operator is overloaded in the derived class, data members exclusive to the derived class will be unassigned
  - Base class has less data members than the derived class
    - Some data members missing in the derived class object

Software Engineering With Inheritance

- Classes are often closely related
  - “Factor out” common attributes and behaviors and place these in a base class
  - Use inheritance to form derived classes
- Modifications to a base class
  - Derived classes do not change as long as the public and protected interfaces are the same
  - Derived classes may need to be recompiled

Composition vs. Inheritance

- "is a" relationship
  - Inheritance
- "has a" relationship
  - Composition - class has an object from another class as a data member
  - `Employee " is a " BirthDate; //Wrong!`
  - `Employee " has a " BirthDate; //Composition`

“Uses A” And “Knows A” Relationships

- “uses a” relationship
  - One object issues a function call to a member function of another object
- “knows a” relationship
  - One object is aware of another
    - Contains a pointer or handle to another object
    - Also called an association

Case Study: Point, Circle, Cylinder

- Define class `Point`
  - Derive `Circle`
  - Derive `Cylinder`
### Point definition

```cpp
// Definition of class Point
class Point
{
  using std::ostream;

public:
  Point( int a, int b ) { setPoint( a, b ); }
  Point() : Point( 0, 0 ) { setPoint( 0, 0 ); }

private:
  int x, y;      // coordinates of the point

protected:
  int getX() const { return x; } // get x coordinate
  int getY() const { return y; }  // get y coordinate

// Constructor for Point
Point( int a, int b ) { setPoint( a, b ); }

// With a member initializer and initializes radius
Circle( double r, int a, int b ) : Point( a, b ) { setRadius( r ); }

// Call base-class constructor
Point( int a, int b ) : Circle( a, b ) { setPoint( a, b ); }

// Constructor for class Point
Point( int = 0, int = 0 );     // default constructor

friend ostream &operator<<( ostream &, const Point & );
```

### Circle definition

```cpp
// Definition of class Circle
class Circle
{
  using std::ostream;

public:
  Circle( double r = 0.0, int x = 0, int y = 0 );
  Circle( double r ); // default constructor

private:
  int x, y;      // coordinates of the circle
  double radius;   // radius of the Circle

protected:
  int getX() const { return x; } // get x coordinate
  int getY() const { return y; } // get y coordinate
  double getRadius() const { return radius; } // return radius
  void setRadius( double ); // set radius

friend ostream &operator<<( ostream &, const Circle & );
```

### Cylinder definition

```cpp
// Definition of class Cylinder
class Cylinder
{
  using std::ostream;

public:
  Cylinder( double h = 0.0, double r = 0.0, int x = 0, int y = 0 );
  Cylinder( double h, double r, int x, int y );
  Cylinder( double h, double r );
  Cylinder( double h ); // default constructor
  Cylinder( double r, int x, int y );
  Cylinder( double r ); // default constructor
  Cylinder( int x, int y ); // default constructor

private:
  int x, y;      // coordinates of the Cylinder
  double height;            // height of the Cylinder

protected:
  double volume() const ;      // calculate and return volume
  double area() const ;        // calculate and return area
  double getHeight() const ;  // return height
  void setHeight( double );  // set height

friend ostream &operator<<( ostream &, const Cylinder & );
```
Example.cpp

Driver
1. Load headers
1.1 Initialize object
2. Function calls
2.1 Change attributes
3. Output
4. Output

Program Output